بسم الله الرحمن الرحيم

جامعة النيل الأزرق

كلية الدراسات الإضافية

أساسيات البرمجة

fundamentals of programming

الجزء الأول

إعداد : أ. عبد الرحمن عباس إبراهيم

2007

مقدمة Introduction:

تتجه كل المؤسسات الحكومية منها و المدنية ، التجارية و العسكرية و المدارس ، حتى في المنازل في المطابخ و الغرف ، تتجه لاستخدام الحاسب الآلي في تنفيذ الأعمال ، وهذا يتطلب وجود برامج تناسب كل مجال من المجالات السابقة ، فمثلاً نحتاج لبرامج لإعداد الوجبات الغذائية المتكاملة . و برامج تعليمية للمدارس و برامج حسابات و مخازن للتجار ، و برامج رسم هندسي للمهندسين ..الخ. كل هذه المهام – مهام تصميم البرامج – تقع على عاتق المبرمجين ، فهم المسئولون عن تصميم البرامج بالشكل الذي يناسب المستخدمين .

البرمجة تعني كتابة البرامج الحاسوبية بأسلوب علمي يضمن حلول حقيقة دقيقة للمسائل البرمجية و باستخدام إحدى لغات البرمجة ، وتمثل لغات البرمجة الأداة الأساسية المستخدمة في كتابة و تنسيق و ترجمة و تنفيذ البرامج .

تعريف النظام System definition :

النظام عبارة عن مجموعة من الوحدات – الأنظمة الفرعية أو النظيمات – التي تتكامل مع بعضها لإنجاز مهام محددة و كل وحدة أو نظام فرعي من وحدات النظام يوكل إليها جزء من المهام .

تعريف الحاسب Computer Definition :

الحاسب عبارة عن جهاز الكتروني يضم أجهزة كهربائية و ميكانيكية و معدات الكترونية يتلقى التعليمات من المستخدم و يقوم بإنجاز العمليات الحسابية و المنطقية ليقدم مخرجات يستفيد منها المستخدم ويقدم حلول و نتائج لدعم القرار.

مكونات نظام الحاسب Computer System Components :

يتكون نظام الحاسب الآلي من ثلاثة مكونات أساسية (تمثل أنظمة فرعية) هي:

المكونات المادية Hardware:

و تمثل الجزء الأساسي من نظام الحاسب الآلي و هي الأجزاء المادية الملموسة من النظام و تصنف إلى جزأين (أنظمة فرعية) :

- **وحدة النظام System Unit.**
- **الوحدات الطرفية Peripheral Units**

- **أولاً وحدة النظام** System Unit :

و هي الوحدة الأساسية في نظام المكونات المادية فهي تضم وحدة المعالجة المركزية المسئولة عن معالجة البيانات و وحدة الذاكرة الرئيسية المسئولة من تخزين البيانات الجاري تنفيذها في وحدة المعالجة المركزية و الذاكرة الثانوية التي تخزن فيها البيانات بشكل مستمر ، إذن مكونات وحدة النظام الأساسية هي:

1. وحدة المعالجة المركزية Central Processing unit (cpu)
2. الذاكرة الرئيسية Main Memory.
3. الذاكرة الثانوية Secondary Memory.

- **الوحدات الطرفية** Peripheral Units :

يعتبر كل جهاز غير وحدة النظام وحدة طرفية ، و تصنف الوحدات الطرفية إلى قسمين رئيسيين :

وحدات الإدخال Input Units :

و هي الوحدات المسئولة عن إدخال البيانات إلى الحاسب و تختلف وحدات الإدخال حسب نوع البيانات المدخلة فمنها ما هو خاص بإدخال البيانات النصية و منها ما هو خاص بإدخال البيانات الصوتية و الصورية ...الخ.

من وحدات الإدخال :

1. لوحة المفاتيح Key Board.
2. الفأرة الالكترونية Mouse.
3. الماسحة الضوئية Scanner.
4. الكاميرات الرقمية Camera.
5. المايكروفون Microphone.
6. الفاكس Fax.
7. المودم Modem

وحدات الإخراج Out Put Units.

و هي الوحدات المسئولة عن عرض النتائج و المخرجات و كذلك تختلف حسب نوع البيانات التي سيتم إخراجها من وحدات الإخراج :

- شاشة العرض Monitor.

- الطابعة Printer.
- السماعات الخارجية desktop Speaker.
- سماعات الأذن headphone Speaker.
- الفاكس Fax.
- انمودم Modem

ملاحظة : بعض الأجهزة تعمل كوحدات إدخال و إخراج مثل الفاكس مودم.

المكونات البرمجية Software:

تضم منظومة البرمجيات الآتي:

- نظم التشغيل Operating System :
- لغات البرمجة Programming Languages.
- البرامج التطبيقية Applications .

أنظمة التشغيل عبارة عن مجموعة برامج تعمل كوسيط بين المستخدم و المكونات المادية ، تمكن المستخدم من استخدامها بسهولة و يسر، كما تمكنه من التحكم فيها و إدارتها ، من أمثلة نظم التشغيل :

- ويندوز Windows .
- لينكس Linux.
- يونكس Unix.
- دوس Dos.
- نوفل نتوير Novel Netware.
- سولارس Solaris .

أما لغات البرمجة فهي وسيلة التخاطب بين الإنسان و الحاسب ، وهي أداة بيد المبرمج يستخدمها لكتابة و تصميم و تنفيذ برامج لحل مشاكله البرمجية و هذه اللغات يمكن تصنيفها إلى:

1- لغة الآلة Machine Language و هي اللغة الوحيدة التي يفهما الحاسب ، و تتكون من أرقام من بين (0,1) و هي تختلف من حاسب لآخر .

2- لغة التجميع Assembly language : و هي لغة تستخدم اختصارات معبرة من اللغة الإنجليزية لتعبر بها عن العمليات الأساسية التي يقوم بها الحاسب من إضافة add و طرح sub و حفظ store و تتعامل مباشرة مع مجموعة مواقع في الذاكرة تسمى المسجلات Register.

3- لغات المستوى الأعلى High Level Language: و هي لغات تستخدم كلمات أقرب إلى لغة الإنسان مثل اللغة الإنجليزية ، هنالك الكثير من هذه اللغات مثل(بيسك basicو باسكال Pascal و فرتران Fortran سي و سي++ c/c++ ، و هنالك لغات أكثر تطوراً و هي لغات Visual مثل visual c++ و visual basic.. الخ.

- أما البرامج التطبيقية فهي برامج صممت بواسطة المبرمجين لحل مشاكل برمجية ، و تضم حزم البرامج الجاهزة ، التي تتولى شركات مثل مايكروسوفت إنتاجها مثل حزمة Office ، و برامج تطبيقات تصمم لحل مسائل برمجية بسيطة بواسطة لغات البرمجة .

المكونات البشرية Heartware :

هم الأشخاص الذين يتعاملون مع نظام المكونات و البرمجيات تختلف مهامهم فمنهم المبرمجون و منهم مهندسو النظم و محللي النظم و مدخلو بيانات و غيرهم.

البرنامج Program:

عبارة عن مجموعة من التعليمات مكتوبة بإحدى لغات البرمجة تعطى للحاسب الآلي ليقوم بعمل ما مثل حساب مجموع قيم رقمية.

المبرمج Programmer :

هو شخص ذو دراية و معرفة تامة بإحدى لغات البرمجة أو مجموعة منها و قادر على تحليل المشاكل البرمجية و تصميم حل مناسب لها باستخدام تلك اللغة أو إحدى تلك اللغات .

البرنامج المصدر Source Program:

هو البرنامج المكتوب بإحدى لغات البرمجة (لغات المستوى الأعلى مثل c/ c++ / /basic Pascal) و يمثل تعليمات برمجية لحل مسألة أو مشكلة ما.

البرنامج الهدف: Object Program:

هو البرنامج الناتج عن ترجمة البرنامج المصدر باستخدام مترجم لغة برمجة Compiler أو مفسر interpreter و يكون مكتوب بلغة الآلة و يمكن تنفيذه للحصول على النتائج .

أساليب البرمجة Programming Methods :

مرت عملية البرمجة بمراحل تطور مختلفة ابتداءاً من البرمجة بلغة الآلة – تتطلب البرمجة بلغة الآلة فهم المكونات المادية للحاسب فهم تام بالإضافة إلى فهم تعليمات لغة الآلة – و حتى البرمجة بلغات البرمجة كائنية التوجه OOP التي جعلت عملية البرمجة سهلة و بسيطة تتطلب فقط معرفة الكائنات و كيفية استخدامها بدلاً عن برمجتها فيما يلي سنوضح أساليب البرمجة المتبعة في كتابة و تصميم البرامج .

- **البرمجة الإجرائية** Procedural Programming .

في أسلوب البرمجة الإجرائية يكتب البرنامج كله كتلة واحدة في ملف واحد ، مما يجعل عملية البرمجة صعبة جداً لتداخل التعليمات و كثرتها فيصعب فهم البرنامج ويصعب معرفة الأخطاء اللغوية و المنطقية . من أمثلة اللغات التي تتبع أسلوب البرمجة الإجرائية إصدارات لغة البيسك الأولى (GW-Basic و BASICA).

- **البرمجة الهيكلية** Structural Programming .

أسلوب البرمجة الهيكلية غير نمط البرمجة الإجرائية بتقسيمه للبرنامج إلى مقاطع صغيرة و يعطي كل مقطع اسم معين و توكل إليه مهمة محددة و عند تنفيذ تلك المهمة يتم استدعاء ذلك المقطع ، هذه المقاطع تعرف بالبرامج الفرعية Sub Routines أو تعرف بالدوال و الإجراءات Functions & Procedures في بعض لغات البرمجة ، هذا التقسيم جعل من السهل فهم البرنامج و معرفة مكان الأخطاء اللغوية و المنطقية . و لكن اذا كبر البرنامج و تعقدت تعليماته و كثرت برامجه الفرعية (الدوال و الإجراءات) يكون من الصعب متابعة البرنامج و فهم تعليماته ، فكان أسلوب البرمجة بالأهداف الموجهة Object Oriented Programming (OOP).أمثلة للغات البرمجة الهيكلية لغة باسكال و لغة السي و لغات الفورتران و الكوبول .

- البرمجة بالأهداف الموجهة Object Oriented Programming :

في أسلوب البرمجة بالأهداف الموجهة (OOP) يتم تقسيم البرنامج إلى وحدات ذاتية الاحتواء تضم البيانات و مجموعة من البرامج الفرعية في كيان ، تسمى هذه الوحدات بالكائنات و كل كائن له صفات و له سلوك يميزه عن الكائنات الأخرى ، و تمثل البرمجة الكائنية عناصر البرنامج تمثيل حقيقي مطابق لتمثيل الكائنات العالم الحقيقي .

فوائد البرمجة بالأهداف الموجهة OOP benefits:

1- التجريد Abstraction (حماية و إخفاء البيانات) : إخفاء تفاصيل تصميم الكائن عن المستخدم أي استخدام الكائن دون الحاجة إلى معرفة تفاصيل تركيبه .

2- الكبسلة Encapsulation: وضع كل من البيانات و العمليات (الدوال) في مكان واحد يساعد المبرمج على التعامل مع الكائن بسهولة مثل نسخه وتعريفه .

3- إعادة الاستخدام Reuse (الوراثة inheritance) : يمكن للمبرمج إعادة استخدام كائن مرة أخري دون الحاجة إلى إعادة بناء الكائن من جديد مما يوفر الجهد و يزيد سرعة إنتاج البرامج ، و يمكن بناء كائن جديد يرث خصائص كائن آخر و يضيف إليها خصائصه.

4- تعدد الأشكال Polymorphism: من خلال تعدد الأشكال يمكن أن نجعل دالة ما تؤدي أكثر من وظيفة اعتماداً على الهدف الذي تتبع له.

المترجم Compiler:

من برامج النظم يقوم بترجمة البرنامج المصدر إلي برنامج بلغة الآلة قابل للتنفيذ ، وتتم ترجمة كل البرنامج دفعة واحدة و لا يتم تنفيذ البرنامج إلا بعد التأكد من خلوه من الأخطاء اللغوية .

المفسر Interpreter:

أيضاً من برامج النظم يقوم بترجمة البرنامج المصدر إلى برنامج بلغة الآلة قابل للتنفيذ ، و يختلف عن المترجم في أنه يقوم بترجمة التعليمات و تنفيذها تعليمة تلو الأخرى .

خطوات حل المسائل البرمجية (البرمجة):

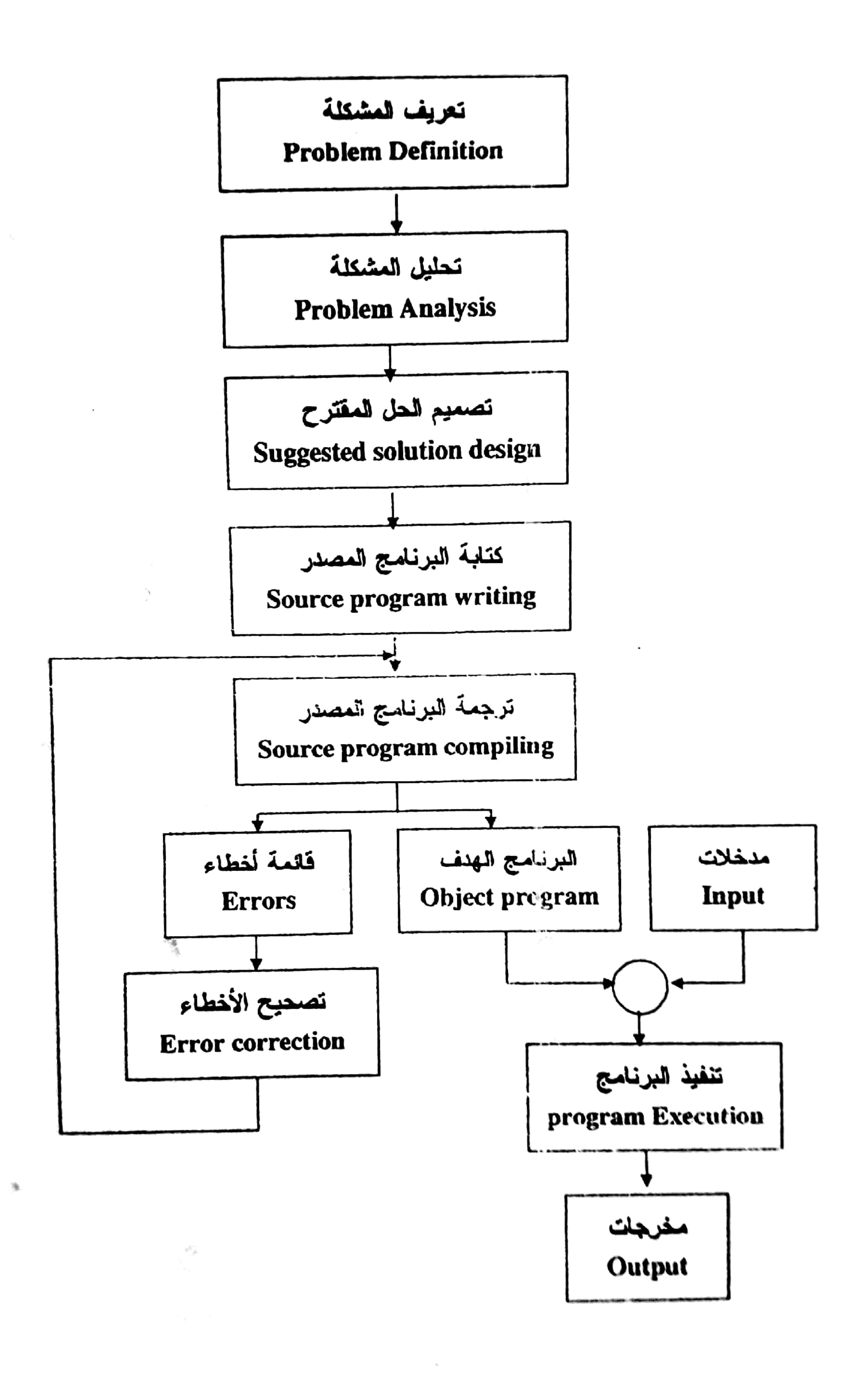
كما ذكرنا سابقا أن البرمجة تعني كتابة برامج باستخدام لغات البرمجة بصورة علمية تقود لحل المسائل البرمجية بصورة سليمة تضمن حلول أكيدة و موثوق بها ، وحتى نحصل على هذه الحلول الوثوق بها لابد من أن تمر عملية البرمجة بعدة مراحل نذكرها فيما يلي بالتفصيل :

1. تعريف المشكلة Problem Definition
2. تحليل المشكلة Problem Analysis.
3. تصميم الحل المقترح solution design.
4. برمجة الحل (كتابة البرنامج) solution Programming.
5. تنفيذ الحل – اختبار البرنامج Solution Implementation.
6. تشغيل البرنامج للحصول على الحلول و النتائج Program Execution.

يمكننا تقسيم الخطوات السالفة الذكر إلي مرحلتين ، الأولى تمثل دور الإنسان في حل المشكلة و الثانية تمثل دور الحاسب في حل المشكلة كالتالي:

- **المرحلة الأولى (دور الإنسان في حل المشكلة) :**
  - تعريف المشكلة .
  - تحليل المشكلة .
  - تصميم الحل المقترح .
- **المرحلة الثانية (دور الحاسب في حل المشكلة) :**
  - برمجة الحل المقترح .
  - تنفيذ الحل _اختبار البرنامج.
  - تشغيل البرنامج .

الشكل التالي (1-1) يبين خطوات حل المشكلة .



خطوات حل المسائل البرمجية

أولاً: تعريف المشكلة.

قبل البدء في حل المسائل البرمجية لابد من تعريف كل مسألة برمجية يراد إيجاد حل لها تعريفاً كاملاً ، و نقصد بتعريف المسألة فهمها فهماً تاماً و تحديد حدودها حتى لا يكون الحل ناقصاً أو غير كافياً أو أن يحيد الحل النهائي عن الحل المطلوب .

الكثير من المشاكل تبدو أكثر تعقيداً عن الحقيقة التي هي عليها و ذلك لعدم فهما فهماً عميقاً ، إذاً في هذه الخطوة يجب على المبرمج فهم المسألة و فهم كل جزيئاتها و كل ما يتعلق بها ، و تقسيمها إلى مشاكل فرعية بسيطة يسهل فهمها إن كانت معقدة.

ثانياً: تحليل المشكل :

و نعني بتحليل المشكلة تحليل المدخلات المطلوبة للمشكلة و معرفة كيفية معالجتها للوصول إلى الحلول المطلوبة و كذلك معرفة شكل المخرجات النهائية التي سيتم عرضها .

- تحليل المدخلات :

لابد من معرفة البيانات التي سيتم إدخالها للبرنامج كمعطيات لحل المشكلة و تحديد نوعها و حجمها مثلاً لإيجاد مجموع ثلاثة أعداد ، المعطيات لهذه المسألة ستكون ثلاثة أعداد يمكن تمثيلها ب X,Y,Z بحيث تمثل هذه المتغيرات أنواع رقمية بأقصى حجم يمكن أن تسمح به لغة البرمجة . إذا لم يتم الحصول على قيم هذه المتغيرات لن يكون هنالك معالجة أو مخرجات و نتائج .

- تحليل المعالجة :

للحصول على المخرجات لابد من معالجة البيانات التي تم إدخالها ، تحليل المعالجة يعني تحديد الطريقة التي سيتم عبرها الحصول على المخرجات ، مثلاً لمعالجة المسألة السابقة (إيجاد مجموع ثلاثة أعداد) فإننا سنستخدم المعادلة التالية لمعالجة المدخلات :

$$Sum=X+Y+Z$$

- تحليل المخرجات :

من خلال تحليل المخرجات سيتم تحديد كيفية عرض المخرجات بشكلها النهائي للمستخدم ، إذ لابد أن توافق المخرجات متطلبات المستخدم . في المسألة السابقة سيتم عرض قيمة المتغير SUM الذي تم حسابه سابقاً.

ثالثاً : تصميم الحل باستخدام الخوارزميات و خرائط التدفق :

هنالك العديد من الأساليب التي يمكن للمبرمج أن يستخدمها ليخطط حله المقترح ، من هذه الأساليب الخوارزميات ALGORITHS و مخططات التدفق FLOWCHARTS و الشفرة الزائفة PSEUDO CODE

تعريف الخوارزمية Algorithm Definition :

الخوارزمية عبارة عن خطوات مرتبة متسلسلة منطقياً تكتب بأي لغة بشرية لها بداية واحدة و نهاية واحدة تعبّر عن خطوات حل مسألة برمجية ، اسمها مشتق من اسم العالم المسلم محمد بن موسى الخوارزمي ، ويختلف حجمها باختلاف المسائل البرمجية ،و باختلاف الأشخاص الذين يقومون بكتابتها، يمكن وضع أكثر من خوارزمية لحل مسألة برمجية واحدة.

تمييز الخوارزميات بالصفات التالية :

1- لها بداية واحدة و نهاية واحدة.

2- مرتبة و متسلسلة منطقياً.

3- واضحة و بسيطة و غير غامضة .

4- توضح خطوات حل مسألة برمجية .

5- تكتب بأي لغة مفهومة .

أمثلة محلولة (1-1):

أكتب خوارزمية لحل المسائل البرمجية التالية :

1- إيجاد الوسط الحسابي لأربعة أعداد.

2- حساب مساحة الدائرة باستخدام $AREA= PIxR^2$

3- تحويل درجة الحرارة من فهرنهايت F إلى مئوي C بالعلاقة

$C=9/5*(F-32)$

الحلول :

أولاً الوسط الحسابي لـ4 أعداد

1- البداية .

2- أدخل أربعة أعداد A,B,C,D.

3- احسب المجموع SUM=A+B+C+D.

4- اجعل AV=SUM/4.

5- اطبع الوسط الحسابي AV.

6- النهاية.

ثانياً مساحة الدائرة :

1- البداية .

2- أدخل نصف القطر R .

3- اجعل PI=3.14.

4- احسب المساحة AREA=PI*R*R .

5- اطبع المساحة AREA.

6- النهاية .

ثالثاً التحويل من فهرنهايت F إلى مئوي C:

1- البداية .

2- أدخل درجة الحرارة بالفهرنهايت F.

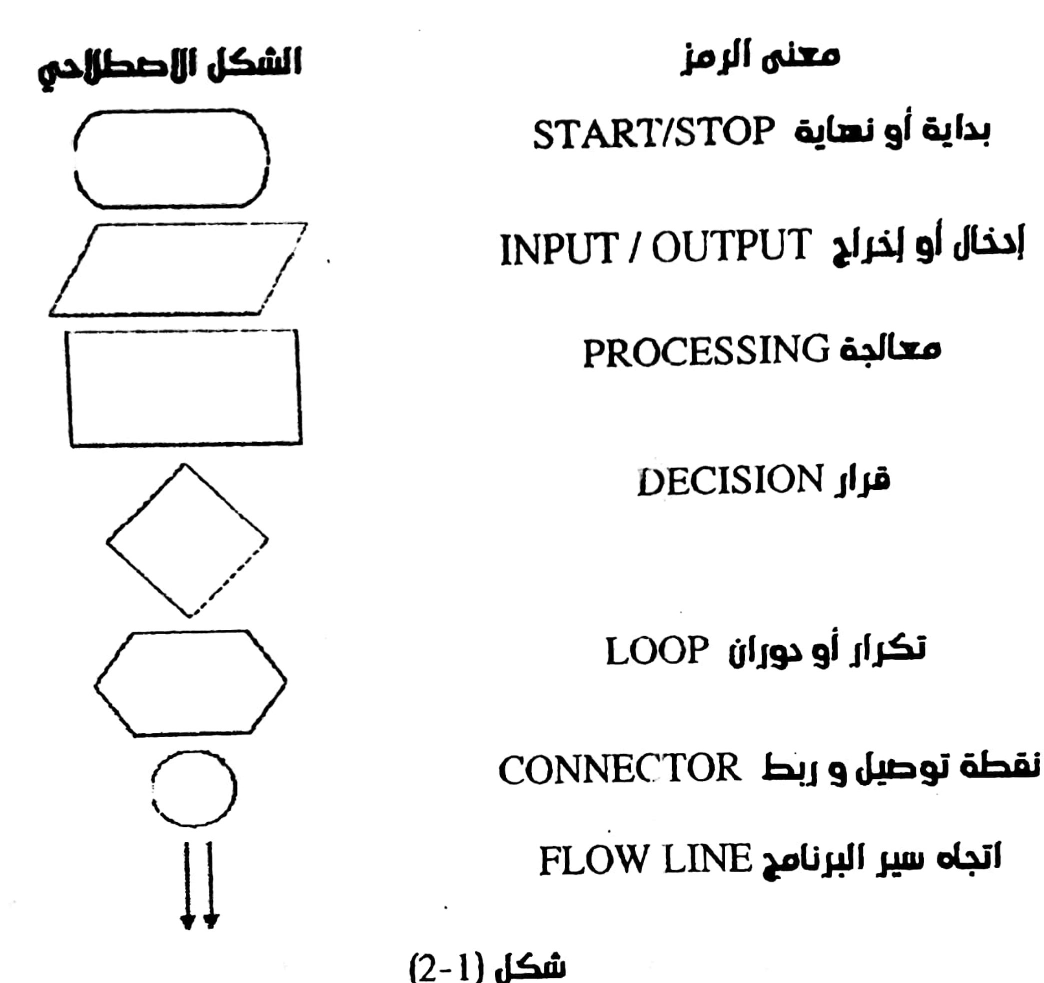
3- اجعل C=9/5*(F-32)

4- اطبع درجة الحرارة بالمئوي C.

5- النهاية .

منخططات التدفق Flow Chart :

تستخدم خرائط التدفق لبيان خطوات حل المشكلة و كيفية ارتباطها ببعضها ، باستخدام رموز اصطلاحية لتوضيح خطوات الحل و هذه الرموز مبينة بالشكل التالي:



شكل (1-2)

من أهم فوائد استخدام خرائط التدفق قبل كتابة البرنامج

1- تعطي صورة متكاملة للخطوات المطلوبة لحل المشكلة .

2- تمكن المبرمج من الاحاطة التامة بكل أجزاء المسألة .

3- تساعد المبرمج على تشخيص الأخطاء ، وخاصة الأخطاء المنطقية.

4- تيسر للمبرمج أمر إدخال أي تعديلات في أي جزء من المسألة.

أنواع خرائط التدفق :

هنالك نوعان رئيسيان من خرائط العمليات :

- **خرائط سير النظم** SYSTEM FLOWCHARTS:

يستخدم هذا النوع من الخرائط عند تصميم الأجهزة الهندسية في المصانع و غيرها و التي تستخدم أنظمة ذاتية التحكم .

- **خرائط سير البرامج** PROGRAMS FLOWCHARTS:

و يستعمل هذا النوع من الخرائط لبيان الخطوات الرئيسية التي توضع لحل مسألة ما و ذلك بشكل رسوم اصطلاحية تبين العلاقات المنطقية بين سائر خطوات الحل .و يمكن تصنيف خرائط سير البرامج إلى ثلاثة أنواع رئيسية :

1. خرائط التتابع البسيطة SIMPLE SEQUENTIAL FLOWCHART
2. الخرائط ذات الفروع BRANCHED FLOWCHARTS.
3. خرائط الدوران LOOP FLOWCHART.

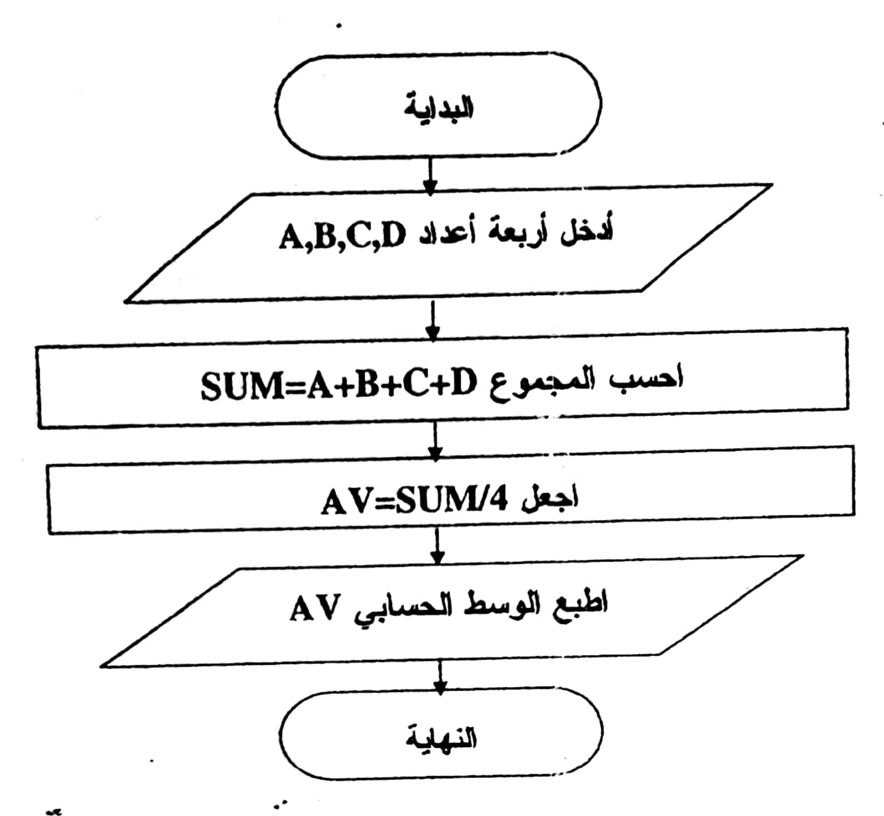
- **أولاً : خرائط التتابع البسيطة :**

في خرائط التتابع البسيطة تكون المسألة بسيطة غير معقدة الخطوات ، و تكون خطوات حلها متسلسلة لا يوجد بها تكرار لعملية ما أو اختيار و تفرع ، مثال لهذه المسائل البرمجية المسائل الثلاثة المذكورة آنفاً ،أدناه أمثلة المخططات التدفقية ذات التتابع البسيط .

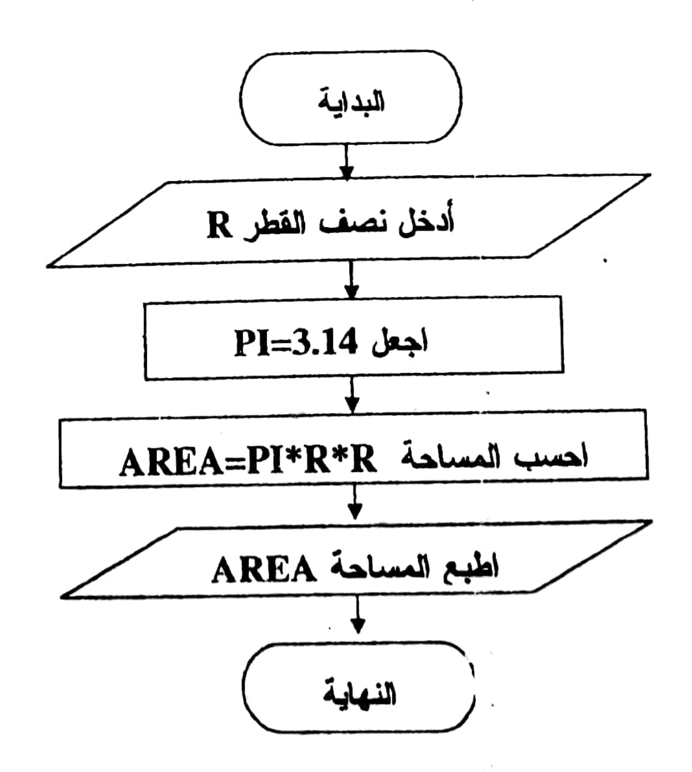
أمثلة محلولة(1-2) : أرسم مخطط التدفق للمسائل في (1-1)

الحلول:

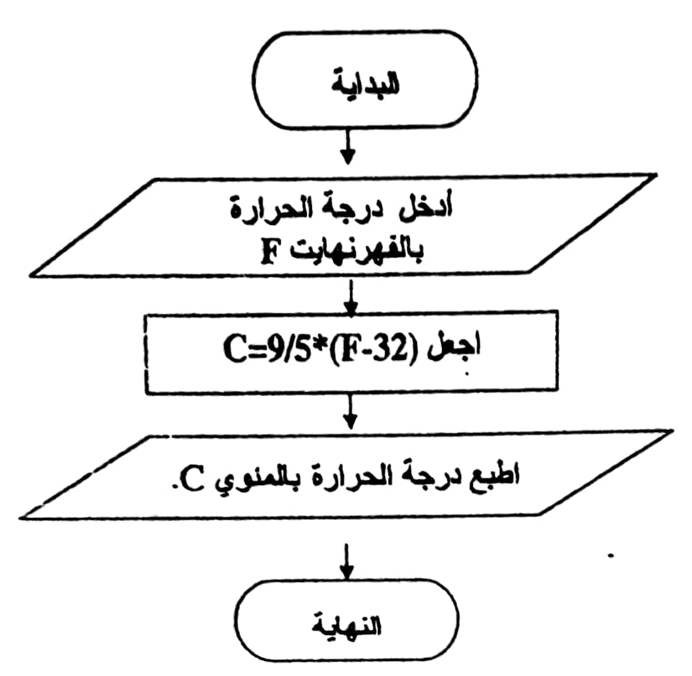
أولاً : الوسط الحسابي لأربعة أعداد :



ثانياً : حساب مساحة الدائرة :



ثالثاً التحويل من فهرنهايت F إلى مئوي C



- **ثانياً :الخرائط ذات الفروع:**

أما المخططات ذات الفروع ، فمثل خرائط لمسائل برمجية معقدة قليلاً، و تحتوي على عمليات تتطلب الاختيار و التفرع المثال التالي يبين شكلاً من هذه الأشكال:

مثال: أرسم مخططاً تدفقياً لمقارنة رقمين و طباعة الرقم الأكبر:

أولاً الخوارزمية :

1- البداية .

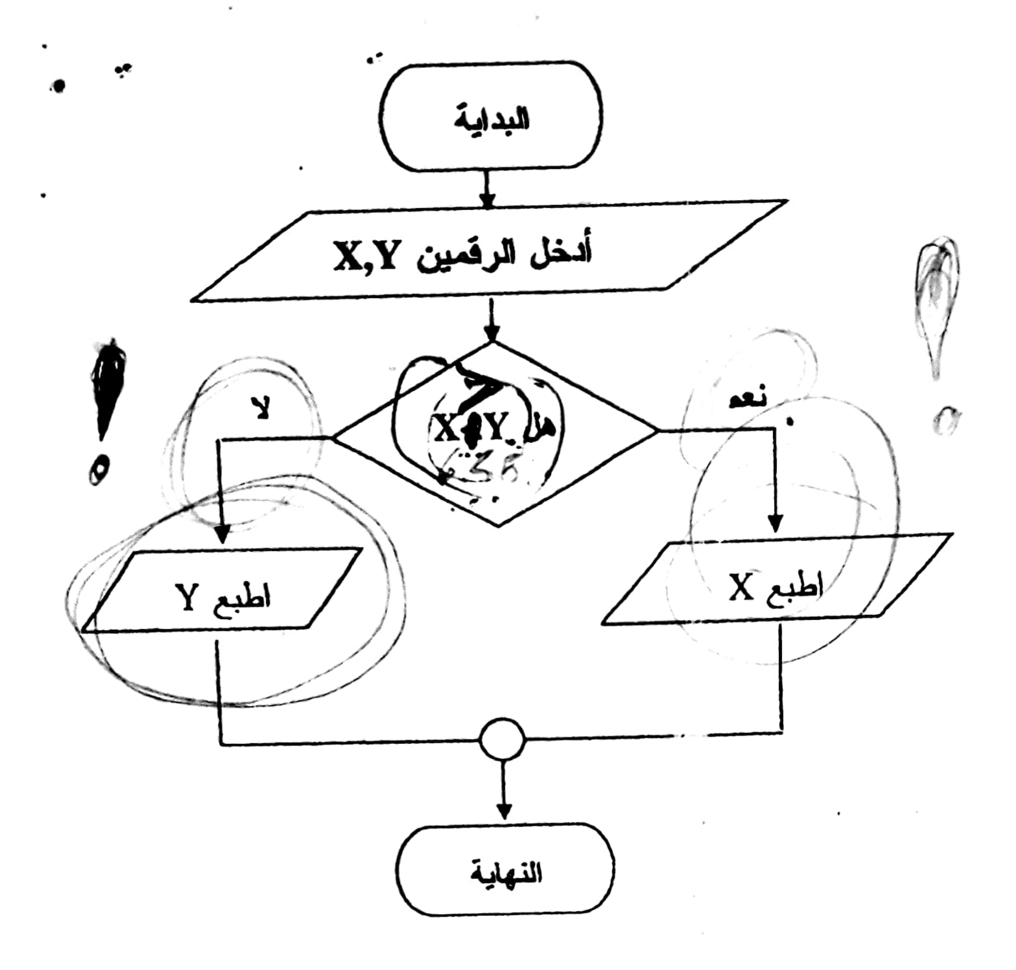
2-- أخل الرقين للمقارنة.

3- هل X>Y .

4- إذا كان الناتج نعم اطبع X ثم اذهب إلى الخطوة 6.

5- اطبع Y.

6- النهاية.



ملاحظة: دائما قبل رسم المخطط الانسيابي لابد من كتابة الخوارزمية لتسهل عليك رسم المخطط.

ثالثاً :خرائط الدوران

بعض المسائل البرمجية تتطلب تكرار عملية معينة عدة مرات ، مثلا ، في مسألة برمجية ما ، نريد أن نكرر تعليمة 100 مرة ، ليس من المنطقي أن نقوم بكتابة 100 خطوة أو رسم 100 خطوة في خريطة التدفق ، و لكن يمكن كتابة خطوة واحدة ثم تكرار هذه الخطوة مائة مرة . المثال التالي يبين هذا النوع من المخططات.

أمثلة محلولة (1-3):

أكتب خوارزمية ثم ارسم خريطة تدفق للمسألة البرمجية التالية:

طباعة الأرقام من 1- 100 على الشاشة بصورة متسلسلة .

الحل:

أولاً الخوارزمية :

1- البداية .

2- اجعل i=1 .

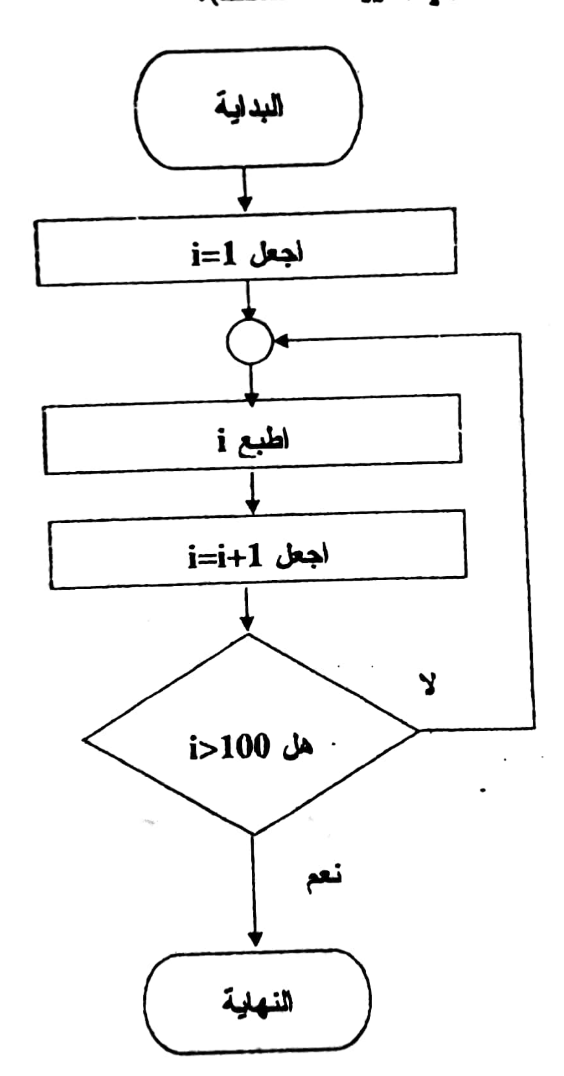
3- اطبع i.

4- اجعل i=i+1.

5- هل I>100 إذا كان لا اذهب إلى الخطوة 3.

6- النهاية.

ثانيا: المخطط الانسيابي (خريطة التدفق):



رابعاً: كتابة البرنامج :

بعد تعريف المشكلة تعريفاً كاملاً و تحديد و تحليل المدخلات و المخرجات ، ثم كتابة الخوارزميات و بناء خرائط التدفق ، يقوم المبرمج بكتابة شفرة البرنامج باستخدام إحدى لغات البرمجة التي يجيدها ، ثم ينقل هذا البرنامج إلى الحاسب ليمثل البرنامج المصدر Source Program ليقوم بترجمته إلى لغة الآلة – البرنامج الهدف Object Program – مستخدماً مترجم اللغة ،خلال عملية الترجمة قد تواجه المبرمج بعض الأخطاء اللغوية – كأخطاء في كتابة تعليمات برمجية – أو أخطاء منطقية – كأخطاء في تسلسل تعليمات البرنامج ، مما يطره إلى تصحيح هذه الأخطاء ، بعدها يصبح البرنامج جاهزاً لتجربته و التحقق من قدرته على إعطاء حلول معقولة و صحيحة منطقياً .

خامساً: تنفيذ البرنامج (اختبار الحل) Solution Implementation:

هذه الخطوة من أهم الخطوات ، فبعد التأكد من خلو البرنامج من الأخطاء المنطقية و اللغوية سيتم اختبار البرنامج بمدخلات بسيطة معلومة القيمة للتأكد من أن البرنامج يعمل بصورة سليمة. و كذلك للتأكد من أهن يعطي الحلول المطلوبة .

سادساً : تشغيل البرنامج بمعطيات حقيقة:

الخطوة الأخير في عملية البرمجة و هي تنفيذ البرنامج باستخدام القيم و المدخلات الحقيقة التي تمثل مدخلات المسالة البرمجة التي من اجلها كتب البرنامج ، يتبع لهذه الخطوة أيضا إضافة التعليقات و العبارات التي من شأنها إزالة اللبس و الغموض عن بعض الجمل البرمجية ، و لمساعدة من يستخدم البرنامج من بعدك في عمليات التعديل و الترقية و الصيانة ، تمسى هذه العملية بالتوثيق .

نهاية الجزء الأول بحمد الله

عماد قنديل _ P

بسم الله الرحمن الرحيم

اساسيات البرمجة بلغة الجافا

الجزء الاول

محمد محمود ابراهيم

جامعة الزعيم الازهري

كلية علوم الحاسوب وتقانة المعلومات

مقدمة : —

تعتبر لغة جافا من اللغات الحديثة جداً في عالم البرمجة، حيث ظهرت بصورة رسمية عام **1990**م وقد قامت بوضع مفاهيمها شركة **Sun Microsystems** . وكان الغرض من ابتكار هذه اللغة برمجة صفحات الإنترنت. انتشرت لغة جافا حول العالم بسرعة كبيرة مع انتشار برمجة صفحات الإنترنت وبرمجة التطبيقات الحديثة الأخرى التي توفرها اللغة مثل برمجة شرائح الهاتف المحمول والبيجر والحواسيب الدفترية وغيرها.

مميزات لغة الجافا : —

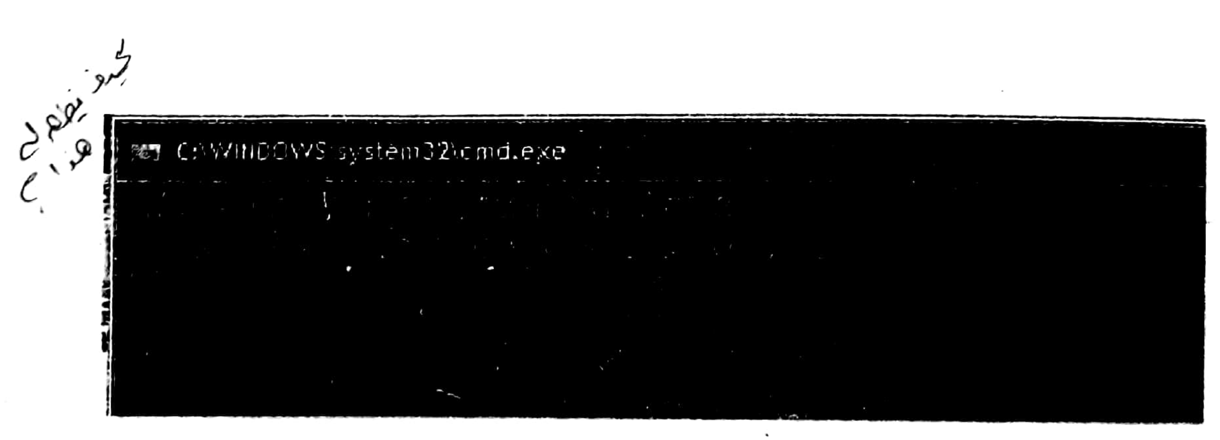
- إنها لغة قوية تحتوي على أدوات كثيرة تساعد في كتابة البرامج.
- لكون جافا لغة حديثة مكنها من تلافي عيوب كثير من اللغات قبلها، من أهم هذه العيوب إمكانية الوصول المباشر لمواقع الذاكرة الخاصة بالبرنامج والذي يؤدي إلى ضعف سرية المعلومات وسهولة تدميرها.
- إن البرنامج المكتوب بلغة جافا يمكن نقله وتشغيله على جهاز حاسوب آخر يحتوي على نظام تشغيل يختلف عن الحاسوب الأول (مثلاً يحتوي **Windows, Linux** وغيرهما) بدون مشاكل.

• تعتبر لغة جافا لغة برمجة بالكائنات (Object Oriented Programming Language) ، ويعتبر هذا الصنف من لغات البرمجة من أوسعها انتشاراً وأكثرها استخدماً اليوم.

لغة جافا كغيرها من لغات البرمجة لاتخلو من العيوب، ويكمن اعتبار لغة جافا بطيئة نسبياً. إن السرعة ميزة مهمة، ولكن يجب التضحية ببعض المميزات لاكتساب مميزات أهم.

وهذا اول برنامج لنتعرف على محتويات برنامج جافا

```
1  class first
2  {
3      public static void main(String args[])
4      {
5          System.out.println(" My First program in Java ");
6
7          }// end of main
8      } // end of class
```



يبداء برنامج جافا بالكلمة المحجوزة **class** يليها اسم البرنامج الذي اختاره المبرمج وهنا **first** ويجب حفظ الملف بنفس الاسم ويحتوي ال **class** على الدالة **public static void main(String args[])** ويبداء تنفيذ البرنامج من هذه الجملة

انواع البيانات في الجافا : –

الاعداد الصحيحة : –

1 byte	**Byte**
2 byte	**short**
4 byte	**int**
8 byte	**long**

الاعداد الحقيقية : –

4 byte	**Float**
8 byte	**Double**

النوع المنطقي : –

Boolean ويشمل القيم **true** او **false**

النوع **String** : –

هذا النوع شائع الاستخدام على الرغم من انه من انواع البيانات غير الاساسية

ويستخدم لتعريف النصوص

المتغيرات : — شروط تسمية المتغير

1. ألا يكون مبدوءاً برقم.
2. ألا يحتوي على فراغات
3. ألا يحتوي على علامات خاصة مثل (-؟!*)
4. ألا يكون من الكلمات المحجوزة مثل

تعريف المتغيرات : —

```
int number1;
int number1 , number2;
double num;
boolean test;
char ch;
String text;
float x;
```

وضع قيمة للمتغير : —

```
float x = 3.7f
number1 = 6;
num = 6.8;
text = "Sudan";
test = true;
ch = 'a';
```

انتبه لهذه أنه بعد تعريف float نكتب f صغيرة بعد الرقم

لاحظ أن في ch وضعنا القيمة بين علامات تنصيص مفردة أما String مزدوجة.

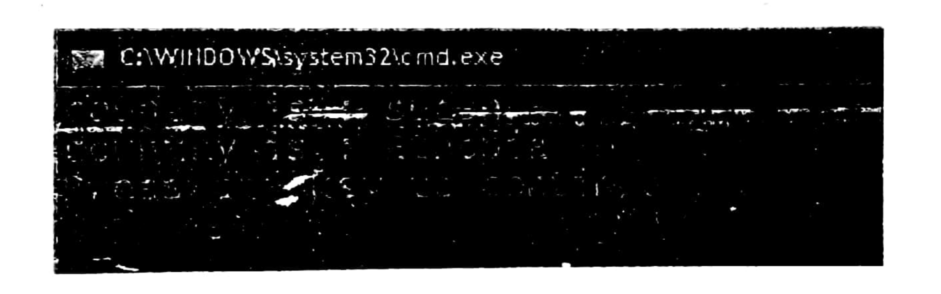
وهذا يعني ان قيمة المتغير **number1** هي 6 و قيمة المتغير **test** هي **true** وهكذا لبقية المتغيرات

مثال : —

```
1  class country
2  {
3       public static void main(String args[])
4       {
5           String count;
6           count = "sudan";
7           System.out.println("country is : " + count);
8           count = "Ethopia";
9           System.out.println("country is : " + count);
10          }
11      }
```

ملاحظة : حاول دائماً أن يكون اسم المتغير له علاقة بقيمته. مثلاً

الخرج من البرنامج



ربط سلاسل نصية :-

لربط السلاسل النصية نستخدم المعامل (+)

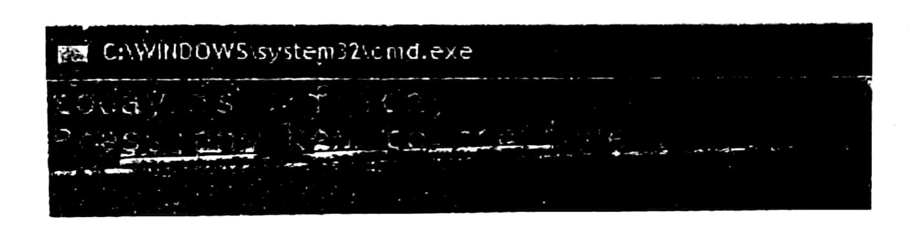
والمثال التالي يوضح ذلك

يمكنك كتابة تعليقات بطريقتين :
1. // في حالة إذا كان التعليق سطراً واحداً
2. /** في حالة إذا كان التعليق أكثر من سطر

مميزات التعليقات :
1. لا يظهر مع المخرجات.
2. يوضح لك الأكواد.
3. في حال كانت الأخطاء كثيرة نضع // على السطر الخطأ لتصبح تعليقاً (لأن البرنامج إذا كان له سطراً خطأ لن يستمر ...)

```
1  class today
2  {
3      public static void main(String args[])
4      {
5
6          String text = "today is : ";
7          String day = "friday";
8          String output = text + day;
9          System.out.println(output);
10
11         }
12     }
```

الخرج من البرنامج



الدالة **print** و **println** تقوم هذه الدوال بعملية الطباعة على الشاشة ولكن الدالة **println** بعد الفراغ من الطباعة تنتقل الي سطر جديد . أو كتابة ("\n") ——— ؟

ملاحظة يمكنك ترك 7 فراغات بواسطة \t و نزول سطر جديد بواسطة \n.

تعريف الثوابت : —

الثابت هو متغير لا يمكن تغير قيمته في البرنامج ولكننا فقط نقوم بتعريفه ووضع قيمة ابتدائية له لحظة التعريف، وتظل هذه القيمة ثابتة طوال البرنامج. تعريف الثوابت لا يختلف عن المتغيرات إلا في الكلمة المحجوزة **final** والتي نكتبها أمام التعريف لنستدل بها على أنه ثابت.

```
final char plus = '+';
final double pi = 3.14;
```

العمليات الرياضية في الجافا : -

a + b	+	Addition الجمع
a − b	−	Subtraction الطرح
a * b	*	Multiplication الضرب
a/b	/	Division القسمة
a % b	%	Modulus باقي القسمة

كيف نكتب الأس في الجافا ؟

العبارات المنطقية : -

العملية	معناها	قيمتها
&&	x and y	صواب فقط إذا كان كل من x و y صواب
||	x or y	خطأ فقط إذا كان كل من x و y خطأ
!	not z	خطأ إذا كان z صواب، وصواب إذا كان z خطأ

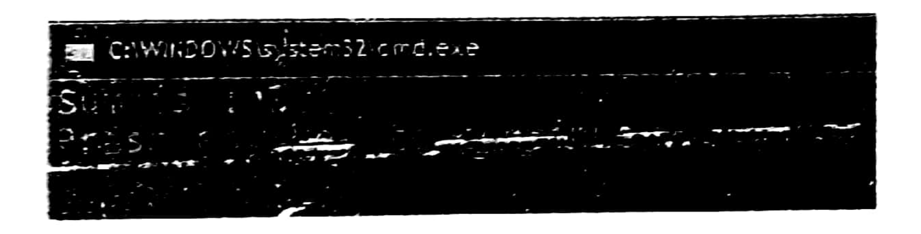
مثال : -

Mohammed.aau@gmail.com

mohammed mahmoud

```
1  class math
2  {
3      public static void main(String args[])
4      {
5          int num1 = 3, num2 = 4;
6          int sum = num1 + num2;
7          System.out.println("Sum is : " + sum);
8
9          }
10     }
```

الخرج من البرنامج



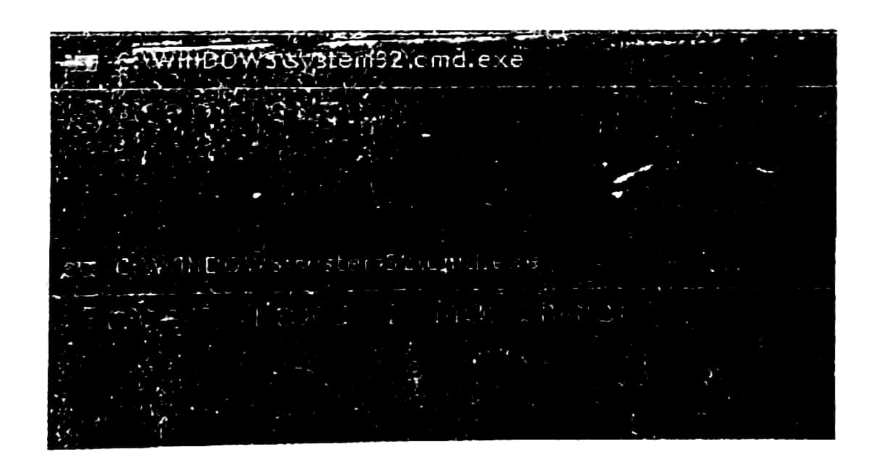
قراءة البيانات من المستخدم : –

لقراءة البيانات من المستخدم نستخدم الكائن **BufferedReaded** الموجود بالحزمة **java.io**

والبرنامج التالي يوضح ذلك

```
1   import java.io.*;
2   // to use BufferedReader
3   class input
4   {
5       public static void main(String args[])throws IOException
6       {
7       String text;
8       char ch;
9       BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
10      System.out.print(" Enter Text : ");
11      text = in.readLine();
12      System.out.print(" Enter Character : ");
13      ch = (char)in.read();
14      System.out.println(text + " - " + cb);
15      }
16  }
```

الخرج من البرنامج



Mohammed.aau@gmail.com

mohammed mahmoud

```
C:\WINDOWS\system32\cmd.exe
Enter Character : M
Press any key to continue . . .
```

طريقة اخرى لقراءة البيانات من المستخدم : —

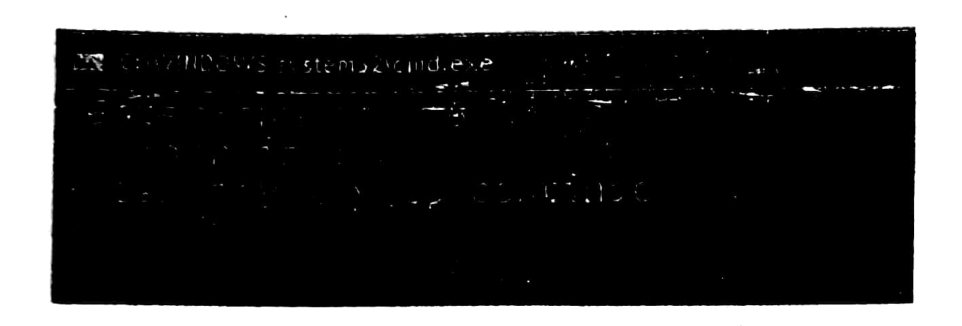
نستخدم الكائن **in** الموجود في الحزمة **System** ولعمل ذلك نستخدم الفئة **Scanner** الموجودة في

import java.util.scanner

والمثال التالي يوضح ذلك

```
1   import java.util.Scanner;
2   // to import scanner class
3   class input
4   {
5       public static void main(String args[])
6       {
7           Scanner in = new Scanner(System.in);
8           System.out.print(" Enter number : ");
9           int m = in.nextInt();
10          System.out.println(" Number is : " + m);
11      }
12  }
```

الخرج من البرنامج



ونلاحظ ان هذه الطريقة اسهل في الادخال

لادخال قيمة من النوع char نستخدم in.nextChar(); و in.nextString() للسلاسل

عبارات المقارنة : -

>	أكبر من
>=	أكبر من أو يساوي
<	أصغر من
<=	أصغر من أو يساوي
==	يساوي
!=	لا يساوي

كيف نقارن نص برقم ؟

جمل الشرط : -

عبارة الشرط if : -

الصيغة العامة لعبارة if

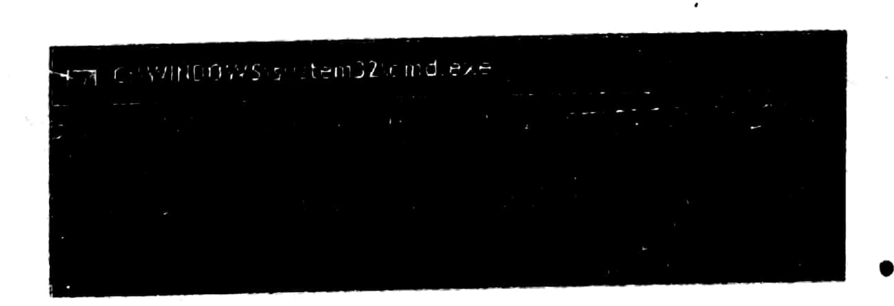
```
if(condition)
{
    statement ;
}
```

والمثال التالي يوضح ذلك

```
1  // if statement program
2  import java.util.*;
3
4  class IfStatement
5  {
6      public static void main(String args[])
7      {
8          int degree;
9          Scanner input = new Scanner(System.in);
10         System.out.print(" Enter your Degree : ");
11         degree = input.nextInt();
12
13         if(degree >= 50)
14         {
15             System.out.println(" Pass ");
16         }
17
18     }
19 }
```

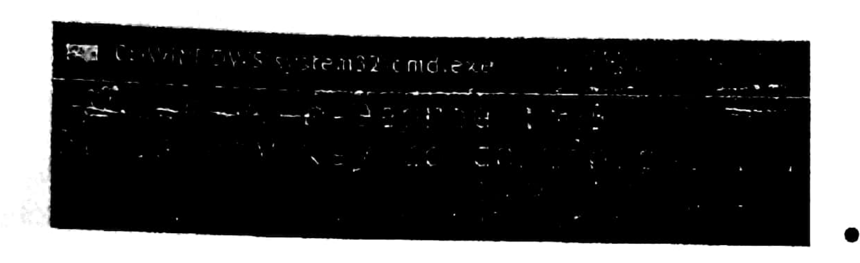
الخرج من البرنامج

- القيمة المدخلة اكبر من 50



-

• القيمة المدخلة اقل من 50



•

العبارة if else : —

```
if(condition)
  statement1 ;
else
  statement2 ;
```

عندما تكون قيمة الشرط **condition** صواباً، يتم تنفيذ **statement1** وتجاهل else والعبارة التي تليها. وعندما يكون الشرط **condition** خطأ يتم تجاهل العبارة **statement1** وتنفيذ العبارة **statement2** . وكما في عبارة **if** ، إذا كان المطلوب تنفيذ أكثر من أمر واحد في حالة قيمة الشرط خطأ، توضع الأوامر بين قوسي بداية ونهاية.

الان سنقوم بتعديل المثال السابق

```java
// if statement program
import java.util.*;

class IfStatement
{
    public static void main(String args[])
    {
        int degree;
        Scanner input = new Scanner(System.in);
        System.out.print(" Enter your Degree : ");
        degree = input.nextInt();

        if(degree >= 50)
            System.out.println(" Pass ");
        else
            System.out.println(" Fail ");
    }
}
```

الخرج من البرنامج

عندما يدخل المستخدم قيمة اكبر من 50



وعندما يدخل قيمة اقل من 50



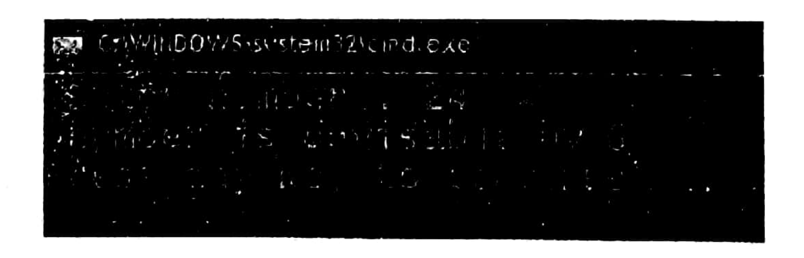
مثال : -

برنامج يحدد اذا كان العدد يقبل القسمة على 6

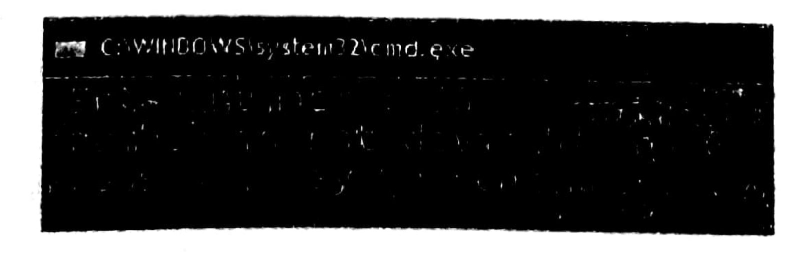
```
1  // program to known number is devisable by six
2  import java.util.*;
3  class test1
4  {
5      public static void main(String args[])
6      {
7          Scanner in = new Scanner(System.in);
8          int number1;
9          System.out.print(" Enter number : ");
10         number1 = in.nextInt();
11         if(((number1%2)== 0)&&((number1%3)==0))
12             System.out.println(" number is devisable by 6 ");
13         else
14             System.out.println(" number is not devisable by 6 ");
15     }
16 }
```

الخرج من البرنامج

عندما ادخل المستخدم الرقم 24



عندما ادخل المستخدم الرقم 56



ويمكن التعبير عن ال if else ب

max = (number1< numner2)?number1:number2;

المثال التالي يوضح ذلك

```
1  import java.util.*;
2  class test2
3  {
4       public static void main(String args[])
5       {
6            Scanner in = new Scanner(System.in);
7            int number1 , number2;
8            System.out.print(" Enter number 1 : ");
9            number1 = in.nextInt();
10           System.out.print(" Enter number 2 : ");
11           number2 = in.nextInt();
12           int max = (number1 > number2) ? number1 : number2;
13           System.out.println(" Maximum number is " + max );
14      }
15 }
```

الخرج من البرنامج



العبارة ...... if else if else : –

```
if(condition)
    statement;
else if(condition)
    statment;
else
    statement;
```

العبارة switch : –

الصيغة العامة للعبارة switch

```
switch(variable)
{
    case value1:
        statement;
    break;
    case value2:
        statement;
    break;
    default:
        statement;
}
```

حيث **variable** هو اسم المتغير المطلوب إجراء الاختبارات على قيمته، ويشترط فيه أن يكون من النوع **int** أو **char** . **value2, value1** عبارة عن قيم يمكن أن يأخذها المتغير.

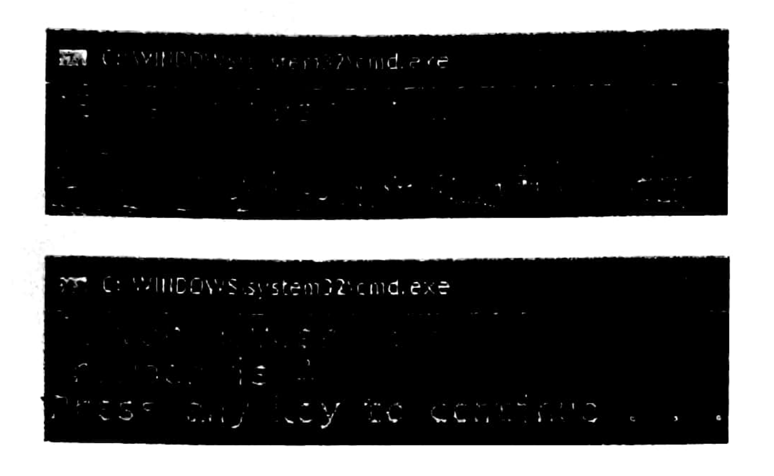
عند إجراء الاختبار على المتغير **variable** ، إذا ساوت قيمته أيّاً من القيم الموجودة بعد كلمة **case** ، يتم تنفيذ العبارة أو العبارات التالية حتى الوصول إلى نهاية **switch** أو العثور على الكلمة **break** ، والتي تقوم بإيقاف تنفيذ عبارات **case** التالية لعبارة **case**

التي تم تنفيذها. أما إذا احتوى المتغير على قيمة غير موجودة ضمن عبارات **case** ، عندئذ يتم تنفيذ العبارات التالية للكلمة المحجوزة **default** .

مثال : -

```
1   import java.util.*;
2   class switchstatement
3   {
4       public static void main(String args[])
5       {
6           Scanner in = new Scanner(System.in);
7           int digit;
8           String number;
9           System.out.print(" Enter number  : ");
10          digit = in.nextInt();
11          switch(digit)
12          {
13              case 1:
14                  number = "one";
15              break;
16              case 2:
17                  number = "two";
18              break;
19              case 3:
20                  number = "three";
21              break;
                default:
                    number = " number is " + digit;
            }
            System.out.println(number);
        }
    }
```

الخرج من البرنامج



الحلقات التكرارية : -

في كثير من البرامج، نحتاج لتكرار تنفيذ جزئية معينة من البرنامج لعدد من المرات، مثلاً إذا كان البرنامج يقوم بقراءة أسماء **50** موظفاً، ليس من المنطقي أن نكتب **50** عبارة قراءة مختلفة. أو إذا كان البرنامج يطبع الأعداد من **1** إلى **1000**، فلا يمكن تصور برنامج يحتوي على **1000** عبارة طباعة، لأنه سيكون طويلاً جداً، وفي نفس الوقت يحتوي على مجموعة من العمليات المتشابهة، وهي عملية الطباعة.

من المتوقع أن نحتاج إلى تكرار تنفيذ العبارات في أغلب البرامج، وخاصة البرامج الكبيرة والأنظمة لأنها تتعامل مع مجموعات من البيانات. ففي نظام للمرتبات، يتم حساب المرتب لكل موظف على حده. أي تكرار عملية حساب المرتب بعدد الموظفين. وفي نظام بنكي، للبحث عن اسم عميل بواسطة رقم حسابه، يتم المرور على جميع عملاء البنك واختبار أرقام الحساب إلى أن نجده أو ينتهي العملاء. ولذلك نجد أن للتكرار أهمية كبرى يكاد لا يستغني عنها أي نظام.

توفر لغة **Java** ثلاث عبارات مختلفة للتكرار. سنتناولها بالتفصيل.

الحلقة while : –

تقوم الحلقة **while** بتكرار العبارات بداخلها مادامت قيمة الشرط **condition** هي **true**

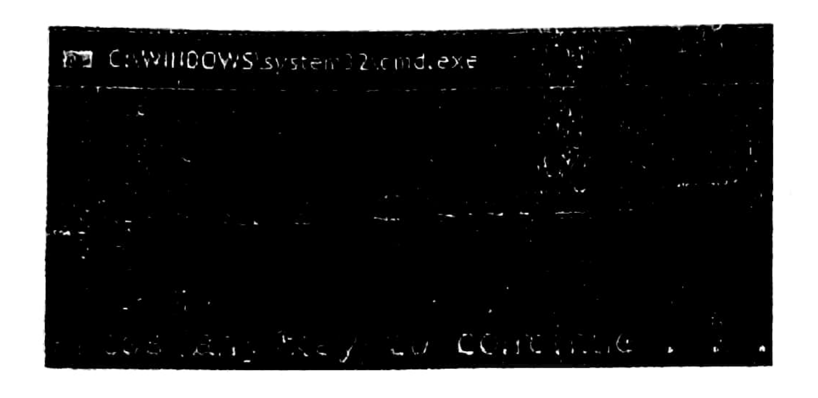
الصيغة العامة للعبارة **while**

```
while(condition)
{
    Statement;
}
```

مثال : –

```
1   // use while statement
2   class whileloop
3   {
4       public static void main(String args[])
5       {
6           int i = 0;
7           while(i <= 5)
8           {
9               System.out.println(" i = " + i);
10              i++;
11          }
12      }
13  }
```

الخرج من البرنامج



الحلقة do while : ـ

هي شبيهة بحلقة **while** إلا أنه يتم اختبار شرطها في نهاية الحلقة. أي أنها تقوم بتنفيذ العبارات الموجودة بداخلها ثم اختبار قيمة الشرط لتحديد استمرارية تكرار عباراتها أو توقفها.

الصيغة العامة للحلقة do while

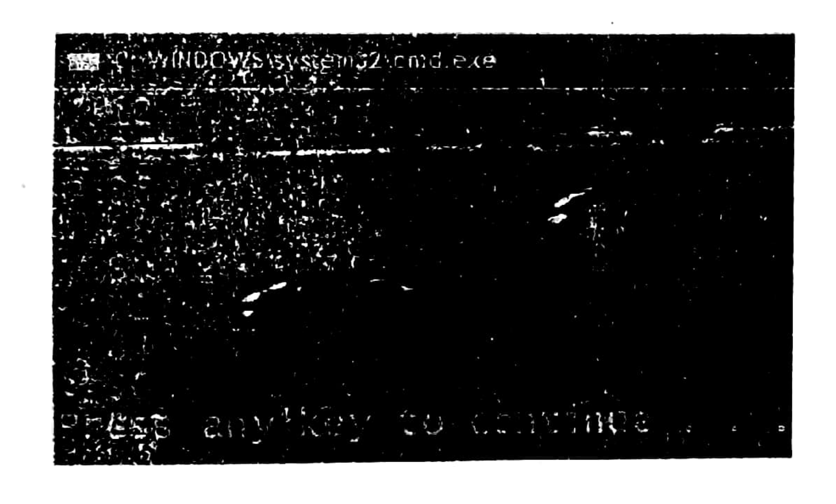
```
do
{
    statement;
}while(condition);
```

مثال : ـ

برنامج يقوم بطباع مضاعفات العدد 12 ال 100

```
1   // use do while statement
2   class dowhileloop
3   {
4       public static void main(String args[])
5       {
6           int i = 12;
7           do
8           {
9               System.out.println("  " + i);
10              i += 12;
11          }while(i <= 100);
12      }
13  }
```

الخرج من البرنامج



الحلقة for : -

عبارة أو حلقة **for** تقوم بتكرار تنفيذ التعليمة **statement** لعدد معلوم من المرات. هذا العدد المعلوم عبارة عن عدد القيم التي يأخذها عداد الحلقة **counter** . يأخذ العداد القيمة الابتدائية **initialValue** ويتم تنفيذ العبارة **statement** ، وبعد كل تنفيذ تزداد قيمة

المتغير counter حسب ما هو معرّف في incrementExpression حتى يصل إلى القيمة النهائية finalValue ، وعندها يتوقف التكرار.

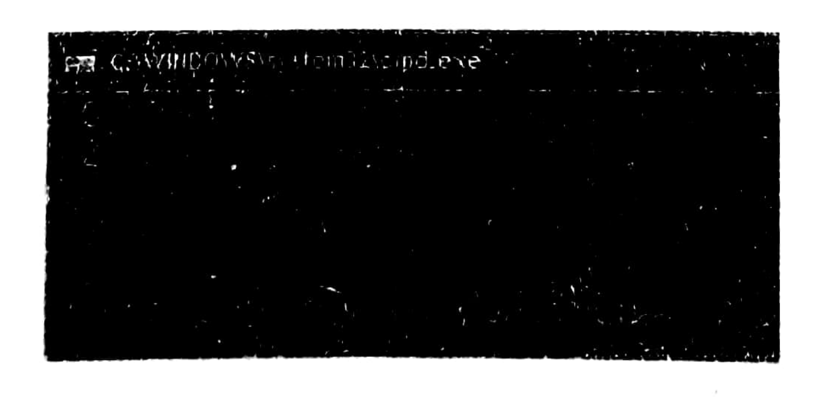
الصيغ العامة للحلقة for

```
for(start;end;frequency)
    statement ;
```

مثال : —

```
1  // use for statement
2  class forloop
3  {
4       public static void main(String args[])
5       {
6           for(int i = 0;i < 10; i+=2)
7               System.out.println(" " + i);
8       }
9  }
```

الخرج من البرنامج

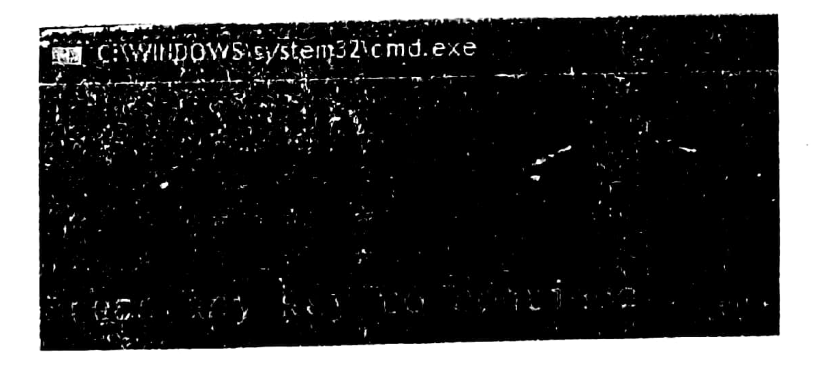


الحلقة for المتداخلة : —

في هذا المثال نستخدم حلقة **for** داخل حلقة **for** اخرى

```
1  // use nested for statement
2  class forloop
3  {
4      public static void main(String args[])
5      {
6          for(int i = 0; i < 5; i++)
7          {
8              for(int j = 0; j < 5; j++)
9                  System.out.print(" * ");
10             System.out.println();
11         }
12     }
13 }
```

الخرج من البرنامج



ملاحظة حول الحلقات التكرارية : –

- عندما نعلم سلفاً عدد التكرارات التي ستنفذها الحلقة، الأفضل استخدام حلقة **for** .

- إذا كنا لا نعلم عدد التكرارات تحديداً، وخصوصاً إذا كان التكرار يعتمد على قيمة يقوم بإدخالها المستخدم، في هذه الحلقة يفضل استخدام **while** أو **do while** .
- إذا كنا نحتاج لعداد لمعرفة رقم التكرار أو استخدام قيمته في البرنامج يمكن استخدام حلقة **for** للاستفادة من عدادها، حيث أن قيمته تبين رقم التكرار.
- إذا كان من الممكن ألا يتم تنفيذ الحلقة أصلاً، فالأصح استخدام حلقة **while** ، أما إن كان تنفيذ الحلقة يكتمل للمرة الأولى في كل الأحوال، يتساوى حينها استخدام **while** و **do while** .
- عموماً عند استخدام لغة **Java** يمكن أن نعبر عن أي فكرة بها تكرار بأيٍ من العبارات التكرارية الثلاث التي توفرها اللغة، وبصورة سليمة وصحيحة، ولكننا دائماً نختار الحلقة الأمثل والأفضل والتي تجعل كتابة البرنامج أسهل وأقل تعقيداً، وتؤدي المطلوب بصورة أكفأ وذلك حسب خواص الحلقة وطبيعة البرنامج المطلوب.

المصفوفات **Arrays** : —

المصفوفة عبارة عن صف من البيانات ذات علاقة ببعضها من نفس نوع البيانات، يكون للمصفوفة اسم واحد وعدد من الحجرات توضع بها البيانات.

المصفوفات احادية البعد : –

الصيغة العامة لتعريف المصفوفة احادية البعد

```
DataType[] VariableName = new DataType[Number];
```

او

```
DataType VariableName[] = new DataType[Number];
```

والمثال التالي يوضح تعريف مصفوفة من النوع **int**

```
int array[] = new int[5];
```

وضع قيم ابتدائية لعناصر المصفوفة : –

الشكل التالي يوضح كيفية وضع قيم اولية للمصفوفة **array**

```
array[0] = 1;
array[1] = 2;
array[2] = 3;
array[3] = 4;
array[4] = 5;
```

تعريف المصفوفة واعطاءها قيم اولية

```
int array[] = new int[]{1,2,3,4,5};
```

طباعة عنصر من المصفوفة

```
System.out.println(array[3]);
```

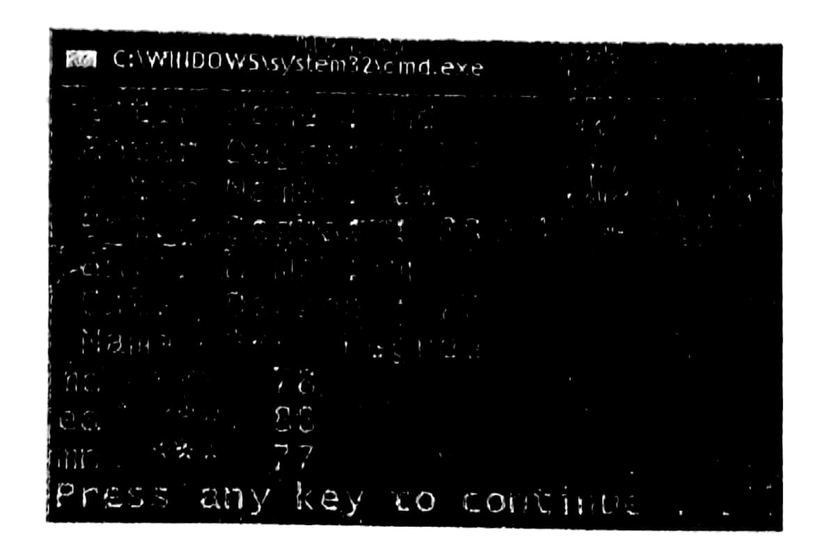
لطباعة كل عناصر المصفوفة نستخدم حلقة for

```
for(int i = 0; i < array.length; i++)
     System.out.print(" " + array[i]);
```

مثال على المصفوفات احدية البعد

```
1   import java.util.Scanner;
2   class student
3   {
4       public static void main(String args[])
5       {
6           Scanner in = new Scanner(System.in);
7           String name[] = new String[3];
8           int degree[] = new int[3];
9           for(int i = 0;i < 3;i++)
10          {
11              System.out.print(" Enter Name : ");
12              name[i] = in.next();
13              System.out.print(" Enter Degree : ");
                degree[i] = in.nextInt();
            }
            System.out.println(" Name " + " *** " + " Degree ");
            for(int j = 0;j < name.length;j++)
            {
                System.out.println(name[j] + "  ***  " + degree[j]);
            }
        }
    }
```

الخرج من البرنامج



المصفوفات متعددة البعد : –

الصيغة العامة لتعريف مصفوفة متعددة البعد

```
int arr2[][] = new int[2][3];
```

تعريف ووضع قيم اولية لمصفوفة متعددة الابعاد

```
int arr2[][] = new int[][] {{1,2,3},{4,5,6}};
```

طباعة عنصر محدد من المصفوفة متعددة الابعاد

```
system.out.print(arr2[0][1]);
```

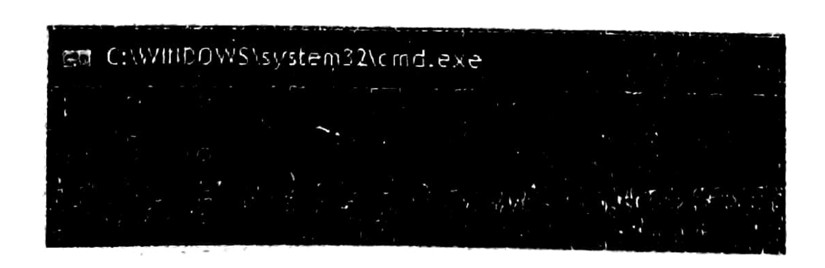
لطباعة كل عناصر المصفوفة نستخدم حلقات for متداخلة

```
for(int i = 0;i < arr2.length;i++)
{
    for(int j = 0;j < arr2[i].length;j++)
    {
        System.out.print(arr2[i][j] + "  ");
    }
    System.out.println();
}
```

مثال على المصفوفة متعددة البعد

```
1  import java.util.*;
2
3  class arraytest1
4  {
5      public static void main(String args[])
6      {
7          int arr2[][] = new int[][] {{1,2,3},{4,5,6}};
8          for(int i = 0;i < arr2.length;i++)
9          {
10             for(int j = 0;j < arr2[i].length;j++)
11             {
12                 System.out.print(arr2[i][j] + "  ");
13             }
14             System.out.println();
15         }
16     }
17 }
```

الخرج من البرنامج



الدوال في الجافا : -

الدالة هي مجموعة من التعليمات التي تؤدي وظيفة معينة، يتم نداؤها خلال البرنامج عند الحاجة بواسطة اسمها. في لغة **Java** تسمى الدوال **methods** . وهناك الكثير من الدوال الموجودة والمعرّفة أصلاً في لغة **Java** والتي يمكن أن نستخدمها عندما نريد. من اكثر دوال لغة **Java** التي استخدمناها خلال الأمثلة الدالة **print** والدالة **println** ، وهما دالتان الغرض منهما الطباعة على الشاشة.

تحتوي لغة **Java** على عدد هائل جداً من الدوال ذات الوظائف المختلفة في شتى المجالات، ولا يتسع المجال لذكرها، بل ولا يمكن حصر جميع الدوال في متناول اليد، ولكن يبحث المبرمج عن الدوال التي يحتاجها بناءً على مجالها. ولاستخدام هذه الدوال لا بده من معرفة طريقة كتابتها ونوع وعدد الوسائط التي تاخذها .

الدوال الجاهزة : -

دوال ال **math class** : -

الطريقة	وصف الطريقة	مثال
`abs(x)`	القيمة المطلقة لـ x.	Math.abs(6.2) → 6.2 Math.abs(-2.4)→ 2.4
`ceil(x)`	تقرب x إلى اقل عدد صحيح ليس اقل من x.	Math.ceil(5.1) → 6 Math.ceil(-5.1)→ -5
`floor(x)`	تقرب x إلى أكبر عدد صحيح ليس أكبر من x.	Math.floor(5.1) → 5 Math.floor(-5.1)→-6
`max(x,y)`	أكبر قيمة من x و y.	Math.max(7,6) → 7
`min(x,y)`	أقل قيمة من x و y.	Math.min(-7,-8) → -8
`pow(x,y)`	x مرفوعة للأس y.	Math.pow(6,2)→6^2 → 36
`sqrt(x)`	الجذر التربيعي لـ x.	Math.sqrt(9)→ $\sqrt{9}$→ 3
`random()`	تكوّن رقم عشوائي بين الصفر والواحد.	Math.random()→ 0.23121

هذه بعض الدوال الموجودة في الفئة math

الدوال الخاصة السلاسل : –

[illegible]	
`s.length()`	ترجع الطريقة `length()` طول السلسلة الرمزية s.
[illegible]	
`s.compareTo(t)`	تقوم الطريقة بمقارنة السلسلة الرمزية s مع السلسلة الرمزية t وتعيد رقم سالب اذا كانت s اقل من t وتعيد صفر إذا كانت s تساوي t وتعيد رقم موجب إذا كانت s أكبر من t.
`s.compareToIgnoreCase(t)`	تعمل هذه الطريقة بنفس عمل الطريقة `compareTo()` ولكن مع اهمال حالة الحروف (صغيرة أم كبيرة).
`s.equals(t)`	تعيد true إذا كان s يساوي t.
`s.equalsIgnoreCase(t)`	تعمل هذه الطريقة بنفس عمل الطريقة `equals()` ولكن مع إهمال حالة الحروف (صغيرة أم كبيرة).
`s.startsWith(t)`	تعيد true إذا كان s يبدأ بالسلسلة الرمزية t.
`s.startsWith(t, i)`	تعيد true إذا كانت السلسلة الرمزية t موجودة في s بدءاً من الموقع i.
`s.endsWith(t)`	تعيد true إذا كان s تنتهي بـ t.

[illegible]	
s.indexOf(t)	ترجع موقع أول مكان توجد فيه t داخل السلسلة الرمزية s.
s.indexOf(t, i)	ترجع موقع أول مكان توجد فيه t داخل السلسلة الرمزية s بعد الموقع i.
s.indexOf(c)	ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s.
s.indexOf(c, i)	ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s بعد الموقع i.
s.lastIndexOf(c)	ترجع موقع آخر مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s.
s.lastIndexOf(t)	ترجع موقع آخر مكان توجد فيه السلسلة الرمزية t داخل السلسلة الرمزية s.

[illegible]	
s.charAt(i)	ترجع الحرف الموجود في الموقع i داخل السلسلة الرمزية s.
s.substring(i)	ترجع جزء من السلسلة الرمزية s بدءاً من الموقع i وحتى النهاية.
s.substring(i, j)	ترجع جزء من السلسلة الرمزية s بدءاً من الموقع i وحتى الموقع j-1.
[illegible]	
s.toLowerCase()	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية s بعد تحويل كل الحروف إلى حروف صغيرة.
s.toUpperCase()	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية s بعد تحويل كل الحروف إلى حروف كبيرة.
s.trim()	إنشاء سلسلة رمزية جديدة من السلسلة الرمزية s بعد الفارغ من البداية والنهاية.
s.replace(c1, c2)	إنشاء سلسلة رمزية جديدة من السلسلة الرمزية s بعد تبديل كل c1 بـ c2، وهما من نوع char.

[illegible]	
s.matches(regexStr)	ترجع هذه الطريقة true إذا كانت السلسلة الرمزية regexStr تطابق السلسلة الرمزية s كاملة.
s.replaceAll(regexStr, t)	إنشاء سلسلة رمزية string جديدة بعد تبديل كل regexStr بـ t.
s.replaceFirst(regexStr, t)	إنشاء سلسلة رمزية string جديدة بعد تبديل أول regexStr بـ t.
s.split(regexStr)	إنشاء مصفوفة تحتوي على أجزاء من السلسلة الرمزية s مقسمة حسب ظهور regexStr.
s.split(regexStr, count)	كما في الطريقة split(regexStr) لكن مع تحديد عدد مرات التقسيم.

الدوال المعرفة بواسطة المستخدم : —

الشكل العام لتعريف الدالة

```
access static return_type method_name(parameters)
{
    statement1;
    statement2;
    .
    .
    .
}
```

Access محدد الوصول ويكون public او private او protected

Static تستخدم لتعريف الدالة ليتم استخدامها داخل الصنف الذي عرفت فيه فقط
.

Return_type يحدد نوع القيم التي تعيدها الدالة

Method_name اسم الدالة

Parameters هي المعاملات . وعند تعريف الدالة تسمى هذه المعاملات بالمعاملات الشكلية.(Formal Parameters) وعند استدعاء الداله تسمى بالمعاملات الفعلية (Actual Parameters)

مثال : –

```
public static void university()
{
    System.out.println(" Alzaim Alazhari University ");
}
```

دالة لا تعيد قيمة ولاتحمل وسائط هذه الدالة تقوم بطباعة النص Alzaim Alazhari University

اشكال الدوال :–

- دالة لا تاخذ وسائط ولا تعيد قيمة
- دالة تاخذ وسائط ولا تعيد قيمة
- دالة تاخذ وسائط وتعيد قيمة

• دالة لا تاخذ وسائط وتعيد قيمة

مثال يوضح اشكال الدوال :-

```
public static void method1()
{
    System.out.println("method 1");
}
public static void method2(String method2)
{
    method2 = "method 2";
    System.out.println(method2);
}
public static int method3(int x)
{
    return x * x;
}
public static int method4()
{
    int x,pi = 3.14;
    return x * pi;
}
```

استدعاء الدوال : -

يتم استدعاء الدالة باسمها كما في الشكل التالي

```
university();
```

مثال : -

```
1   class method
2   {
3       static int sum(int a, int b)
4       {
5           return a * b ;
6       }
7       public static void main(String[] args)
8       {
9
10          System.out.println(" Sumation is " + sum(3,4));
11      }
12  }
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
 Sumation is 12
Press any key to continue
```

النداء الذاتي : –

هو ان تقوم الدالة باستدعاء نفسها بنفسها . البرنامج التالي يقوم بايجاد مضروب العدد **n** باستخدام النداء الذاتي .

```
1   import java.util.*;
2   class recursion
3   {
4       static int fact(int a)
5       {
6           if(a == 1 || a == 0)
7               return 1;
8           else
9               return a * fact(a - 1);
10      }
11      public static void main(String args[])
12      {
13          Scanner in = new Scanner(System.in);
14          int num;
15          System.out.print(" Enter Number : ");
16          num = in.nextInt();
17          System.out.println(" Factorial : " + fact(num));
18      }
19  }
20
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
```

ملاحظة : -

- عند استخدام النداء الذاتي يجب الانتباه إلى ضرورة وجود شرط معين لإيقاف النداء الذاتي، وإلا ستتواصل النداءات لعدد لانهائي من المرات، وعندها لا يتوقف البرنامج عن التنفيذ .

- عند استخدام النداء الذاتي يجب الاحتراس والتأكد من وجود شرط توقف النداءات. لكن الأفضل استبداله بالحلقات لأن تنفيذ البرنامج بالنداء الذاتي يستغرق زمناً أطول في التنفيذ ويستهلك ذاكرة أكبر من تنفيذ نفس البرنامج باستخدام الحلقات.

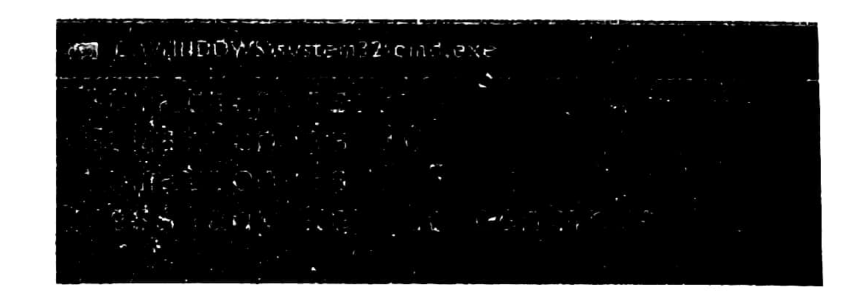
تحميل الدوال بشكل زائد : -

تتم عملية تحميل الدوال بشكل زائد عندما تكون هناك اكثر من دالة تحمل نفس الاسم في نفس الفئة ويتم التميز بين هذه الدوال من خلال عدد المعاملات التي تحملها وانواعها

مثال : -

```
1  class method
2  {
3      static int sum(int a, int b)
4      {
5          return a + b ;
6      }
7      static int sum(int a,int b,int c)
8      {
9          return a + b + c;
10     }
11     static double sum(double a,double b)
12     {
13         return a + b;
14     }
15     public static void main(String[] args)
16     {
17
18         System.out.println(" Sumation is " + sum(3,4));
19         System.out.println(" Sumation is " + sum(3,4,3));
20         System.out.println(" Sumation is " + sum(3.2,4.3));
21     }
22 }
```

الخرج من البرنامج



المراجع : -

- Java how to program 7th edition
- البرمجة بلغة جافا - جامعة السودان المفتوحة